DCAP: Detecting Misbehaving Flows via Collaborative Aggregate Policing

Chen-Nee Chuah
ECE Department
University of California
Davis, CA 95616, USA
chuah@ece.ucdavis.edu

Lakshminarayanan Subramanian
EECS Department
University of California
Berkeley, CA 94720, USA
lakme@cs.berkeley.edu

Randy H. Katz
EECS Department
University of California
Berkeley, CA 94720, USA
randy@cs.berkeley.edu

ABSTRACT

This paper proposes a detection mechanism called *DCAP* for a network provider to monitor incoming traffic and identify misbehaving flows without having to keep per-flow accounting at any of its routers. Misbehaving flows refer to flows that exceed their stipulated bandwidth limit. Through collaborative aggregate policing at both ingress and egress nodes, DCAP is able to quickly narrow the search to a candidate group that contains the misbehaving flows, and eventually identify the individual culprits. In comparison to per-flow policing, the amount of state maintained at an edge router is reduced from $O(n)$ to $O(\sqrt{n})$, where n is the number of admitted flows. Simulation results show that DCAP can successfully detect a majority (64-83%) of the misbehaving flows with almost zero false alarms. Packet losses suffered by innocent flows due to undetected misbehaving activity are insignificant (0.02-0.9%). We also successfully build a prototype that demonstrates how DCAP can be deployed with minimal processing overhead in a soft-QoS architecture.

Keywords

Misbehaving flow detection, Traffic policing, Flow-level accounting

1. INTRODUCTION

Considerable research has focused on extending the Internet architecture beyond best-effort to provide different classes of services to different applications depending on their Quality of Service (QoS) requirements. Existing proposals range from per-flow mechanisms such as IntServ [11] and RSVP [8] to per-class mechanisms such as Diff-Serv [7] and Clearing Houses [10].

Two integral components of any QoS architecture are: admission control and traffic policing [11]. These two components, in combination with appropriate QoS scheduling policies, enable a network provider to dynamically allocate its shared resources to various customers and satisfy their QoS requirements. While admission control limits the number of flows in the system to avoid depletion of resources, traffic policing is responsible for ensuring that the admitted flows use only their allocated share of network resources.

In this paper, we focus on the traffic policing component and address the question of how a network provider can effectively detect *misbehaving* flows with minimal overhead. We define a flow to be *misbehaving* if it generates traffic in excess to its allocated share. It is crucial to detect and penalize misbehaving flows because they can potentially starve other flows sharing the same physical resources, resulting in degraded performance for legitimate flows. We make two simple assumptions: First, the only resource under contention between the flows is the network bandwidth. Second, we use a predictive service model where a source specifies its bandwidth requirement (during admission control) based on its average rate.

The traditional approach to traffic policing in the context of IntServ, Diff-Serv and ATM networks, is to monitor every admitted flow at the routers [9, 15, 35]. Per-flow policing may incur significant processing overhead at the routers ($O(n)$ where n is the number of flows) resulting in poor scalability. To partially alleviate the problem, the policing can be completely shifted to the edge of an ISP's network.

In this paper, we present an alternative to per-flow policing which trades off detection accuracy for increased scalability. Detection accuracy refers to the probability of detecting misbehaving flows at a router. We propose an aggregate policing mechanism, called DCAP (Detection via Collaborative Aggregate Policing), that has both a good misbehaving flow detection probability and a reduced state and overhead at the routers. DCAP works well under the assumption that the number of misbehaving flows is small compared to the total number of flows in the system. We thereby discount the state required for containing misbehaving flows after they are detected from our analysis.

1.1 Paper Contributions and Overview

In this paper, we propose a detection system, Detection via Collaborative Aggregate Policing (DCAP), that allows a network provider[1] to continuously monitor admitted traffic and detect misbehaving flows in an efficient and scalable manner. The design of DCAP is

[1] A network provider refers to a backbone or regional Internet Service Provider (ISP) that administers its own network domain and provides Internet access to individual and corporate customers or smaller service providers.

driven by three goals: (a) to protect the well-behaved flows against resource depletion due to misbehaving flows, (b) to identify and eventually penalize the misbehaving flows, and (c) to achieve robustness with respect to noise conditions and errors in workload modeling.

Our main contribution is to show how one can leverage distributed, aggregate policing at edge routers to quickly identify a *group* that contains the misbehaving flows. The problem can then be simplified into measuring only the flows within this "candidate" group. The principal observation is that it is relatively harmless to have delay in detecting a misbehaving flow when the overall load is relatively low due to statistical multiplexing. This motivates our approach to aggregate multiple flows for group policing, which eliminates per-flow state management at edge routers. We propose an explicit Flow-Identifier (Fid) assignment scheme to group the admitted flows and police the aggregate group. The fact that each edge router maintains only the aggregate state for each *group* is crucial for the reduction of state from $O(n)$, which would be required if each flow were policed individually, to $O(\sqrt{n})$, where n is the number of admitted flows. Aggregate policing also reduces the per-packet processing overhead. In addition, aggregate policing is more robust to noise conditions and variations of a flow's rate. This is because, the aggregate rate of flows in a group has a much lesser variance (as a fraction of the net aggregate rate) in comparison to the average variance of the rate of a flow (as a fraction of the flow's rate) in the group.

We analyze the performance of DCAP through simulations and characterize the trade-offs between different performance criteria by tuning various parameters of DCAP. Results show that DCAP can detect a majority of misbehaving flows with close to zero false-alarms and low detection time for a variety of source models. We also demonstrate the practicality of our scheme by prototyping our algorithm using a Click router [25].

The rest of the paper is organized as follows. In Section 2, we formulate the tracking of misbehaving flow as an on-line change detection problem, in which one needs to detect the occurrence of misbehaving behavior as soon as possible, but with a low rate of false alarms. We discuss related work in Section 2.2. Section 3 describes the Fid assignment, aggregate policing and misbehaving flow detection scheme within DCAP in details. In Section 4, we illustrate how DCAP can be applied to three different scenarios: (a) within a single ISP, (b) in an overlay network, and (c) across multiple ISPs. We explain our evaluation methodology and present our simulations results in Section 5. Section 6 illustrates the operation of DCAP through a proof-of-concept prototype built on top of a Click router [25]. The prototype demonstrates that the processing overhead introduced by DCAP at an edge router is insignificant. Section 7 summarizes key results and addresses several deployment issues.

2. MISBEHAVING FLOW DETECTION

During admission control, a service contract is negotiated between the service provider and the user. The service contract describes the type and amount of traffic sent by the user, and the expected performance offered by the network provider. A key component required to enforce service contracts is a mechanism to detect misbehaving flows that fail to comply with the allocated rate specified in their service contracts. In this section, we will formulate the misbehaving flow detection problem and discuss previous approaches to this problem.

2.1 Problem Formulation

Misbehaving flow detection (MFD) is an example of on-line change detection problems [6], in which one needs to detect the occurrence of abnormal traffic behavior as soon as possible, with a set of constraints, e.g., without exceeding the tolerable number of false alarms. A *false alarm* happens when a flow is detected as misbehaving when it is not. Let n be the number of incoming flows $f_i(s,d)$ between a specific ingress-egress pair (s,d), each with an allocated rate of $A_i(s,d)$, and m be the number of non-complying flows, where $m \leq n$. The problem is to correctly identify as many of these m flows as possible, i.e., to maximize the probability of *successful detection*. The three intuitive performance metrics for evaluating an MFD scheme are:

1. probability of successful detection, $P_{\rm sd}$ (i.e., a misbehaving flow is not detected),
2. probability of false alarms, $P_{\rm fa}$ (i.e., a flow is detected as misbehaving when it is not), and
3. time to detect, $t_{\rm delay}$

Since the main goal of service contract enforcement is to deliver end-to-end QoS assurance to all admitted flows, the most important criteria is to protect the well-behaved flows that use legitimate amount of resources against misbehaving behavior. We quantify how well the performance of well-behaved flows is preserved in terms of $P_{\rm mis}$, the probability of dropped packets from the complying flows. Identifying the misbehaving flow itself is secondary, as long as the impact from undetected flows' activities on other well-behaved flows, i.e., $P_{\rm mis}$, is negligible. When the overall load is relatively low, misbehaving flows can be harmless even if they are not identified. On the other hand, false alarms may seriously degrade the performance of good flows. Therefore, we can tolerate low $P_{\rm sd}$ but $P_{\rm fa}$ should be close to zero.

In summary, an ideal MFD algorithm with the goal of enforcing service contract should achieve the following (in the order of importance):

1. minimum $P_{\rm fa}$,
2. minimum $P_{\rm mis}$,
3. maximum possible successful detection probability, $P_{\rm sd}$, and
4. small $t_{\rm delay}$,

2.2 Previous Approaches

We make the distinction between two different types of traffic policing:

- *profile-based:* The traffic profile of all the flows being policed is known to the router implementing the policing mechanism.
- *profile-less:* The router has no knowledge of the traffic characteristics of the individual flows.

Our problem falls under the domain of profile-based policing since we have prior knowledge of the allocated rates of the individual flows. We will briefly describe the associated related work for both types of traffic policing.

2.2.1 Profile-based policing

The *traffic profile* of a flow describes the characteristics of the traffic generated by the flow (e.g. peak rate, average rate etc.). Any

profile-based policing mechanism is associated with the goal of identifying flows that violate their specified traffic profiles. profile-based policing is normally used in the context of QoS architecture to detect malicious flows (flows that transmit more than their reserved rate as specified in their traffic profile). The most common example of *profile-based* policing is Token Bucket Filter (TBF) [9], a per-flow policing algorithm which ensures that every flow adheres to its traffic profile. Per-flow policing has been used in the context of IntServ [31], DiffServ [15, 35] and ATM [4, 29, 9] networks. While per-flow policing provides the most accurate accounting information, the required state complexity maintained at each edge router grows linearly with the number of flows, n. Maintaining a counter to measure the traffic sent by millions of concurrent flows in the future will be too expensive (using SRAM) or slow (using DRAM) [14]. Current state of the art policing tools like Cisco's Netflow [1] periodically sample packets and may be too slow, inaccurate and memory intensive. Hence per-flow state maintenance may hinder scalability in future high-bandwidth networks.

Machiraju et al. [26] have suggested an alternative approach where the traffic profile is maintained in the packets rather than the routers themselves. This approach uses the Dynamic Packet State(DPS) concept proposed in [33]. Along with the profile, the service provider also provides a certificate during the admission control phase that acts as a proof to ensure that the end-host does not cheat by modifying its traffic profile. Though this approach removes the necessity of maintaining per-flow state at the routers, it does require a modification of the behavior of end-hosts and the packet header format.

2.2.2 Profile-less policing

Profile-less policing attempts to achieve fairness amongst flows traversing a router and distributes the available bandwidth fairly amongst the individual flows. The most common examples of profile-less policing are the fair queuing algorithms. While many of these algorithms like Weighted Fair Queuing (WFQ) [12, 27, 28] maintain per-flow state at the individual routers to enforce fairness, active queue management techniques like Stochastic Fair Blue (SFB) [17] achieve approximate fairness using minimal per-flow state. SFB is derived from a queue management technique BLUE [18] proposed as an alternative to the RED queuing discipline for early detection of congestion. SFB provides a scalable way to identify and rate-limit non-responsive flows using BLUE, which marks or drops packets based on loss rate and link utilization history.

Estan et al. [14] use a SFB-like approach to identify and provide accurate traffic statistics for the heavy-hitters, i.e., elephant flows that contribute to most the majority of the traffic on a specific router interface. Though we would classify this work under profile-less policing, this technique could be useful for detecting malicious flows that are also heavy-hitters and consume a lot of extra bandwidth.

3. OUR APPROACH: DCAP

In this section, we will describe our solution, Detection via Collaborative Aggregate Policing (DCAP), for the misbehaving flow detection problem.

To reduce the processing and state complexity of per-flow policing, we propose to aggregate flows for group policing. In this process, we do sacrifice a certain level of accuracy in tracking flows' behavior. Aggregate policing allows us to detect a *group* that contains the misbehaving flows. The problem can then be simplified into measuring only the flows within this smaller "candidate" group. However, we still need to preserve the uniqueness of each flow to be able to identify a misbehaving flow eventually. To achieve this, each admitted flow is assigned a flow-identifier, Fid, which is then inserted in the packet header. We assume an application proxy on the client side will be configured to insert the proper Fids before forwarding the packets to the edge router.

A few questions remain to be addressed:

1. How are flows assigned into groups?
2. Can an Fid be devised to represent both the flow itself and the groups it is in?
3. How does one combine various aggregate policers to effectively identify misbehaving flows?

In the following two sections, we describe the mechanisms of our scheme, DCAP, that address theses issues.

3.1 Fid Assignment

There are several possible ways to identify flows, and assign them into different bins for aggregate policing. One simple way is to assign each flow with a random identifier, Fid, which does not require a central database to keep track of which Fids have been assigned. This approach comes with two disadvantages. First, potential Fid collisions (if the number of unique Fids is small) will make it impossible to identify unique flow. Secondly, there is a lack of control and flexibility over how flows are aggregated together. Ideally, we would like to assign flows with similar bandwidth requirements and characteristics into the same group for policing.

3.1.1 Fid Assignment and Releasing in DCAP

We introduce the notion of a *Resource Manager* that explicitly assigns $Fids$ to flows. The Resource Manager (RM) is a logical unit that can be physically placed at the fault monitoring point or policy server in an ISP[2]. The RM maintains the repository of admitted flows, their traffic profiles and service contracts for accounting purposes. Only aggregate states of the reservations are maintained at the edge routers (to both reduce the state and the overhead of policing).

We make two observations. First, though we maintain per-flow state at the RM, this state is accessed infrequently for verification purposes. Hence, the communication between the edge routers and the RM is relatively infrequent. Second, the RM, a single logical unit, in practice is implemented in a distributed fashion across different POPs in an ISP (refer to Section 3.3.1 for more details). Therefore the RM is not a single point of failure.

Every Fid consists of two 16-bit sub-fields: $FidIn$ and $FidEg$. All flows that enter or exit at a particular ER are aggregated into different groups based on their $Fids$. The $FidIn$ and $FidEg$ subfields of a flow identify the groups that the flow belongs to at its ingress and egress ER, respectively. In other words, each of the subfields ($FidIn$ and $FidEg$) serves as group identifiers, while together they form a Fid that uniquely identifies a single flow. Before an Fid is allocated to a flow, the RM has to check whether the Fid has been previously assigned to avoid accidental re-use of the same Fid.

[2]The Resource Manager (RM) should be located very close to the collection of edge routers within a given point of presence(POP). The RM can be implemented as in a distributed manner such that every POP is associated with a *local RM* (Section 3.3.1).

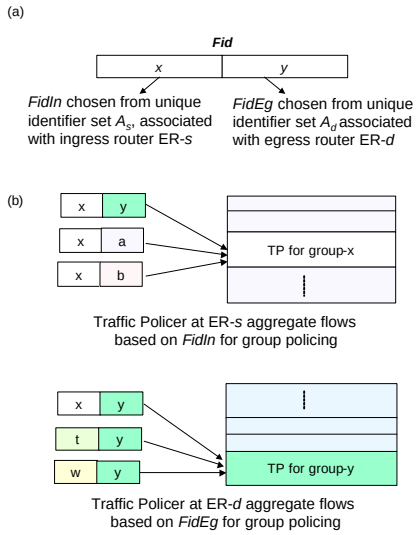

Figure 1: Illustrations: (a) Fid assignments, and (b) aggregate policing at ingress and egress routers.

Each edge router ER_i a set of M unique group identifiers, denoted as $\mathcal{A}_i = \{x_{i1}, x_{i2}, \ldots, x_{iM}\}$, where each member is a 16-bit binary number and is unique across the set $\mathcal{A}_i$. The sets $\mathcal{A}_i$ and $\mathcal{A}_j$ associated with any two ER_i and ER_j are mutually disjoint.

Any Fid of an admitted flow should satisfy the following properties:

1. If the flow is routed from ER_s to ER_d, then $FidIn \in \mathcal{A}_s$, and $FidEg \in \mathcal{A}_d$.
2. No other flow should have the same Fid.
3. Flows with the same $FidIn$ (or $FidEg$) have similar bandwidth requirements.

When a new flow from ER_s to ER_d is admitted, the RM picks a random x_s from $\mathcal{A}_s$ and a random y_d from $\mathcal{A}_d$ such that the Fid with $FidIn = x_s$ and $FidEg = y_d$ satisfies the above properties. This Fid is assigned to the flow, and a new entry with this Fid and its allocated bandwidth is added to the Fid-Repository (FR). The total number of flows that can be uniquely identified in this scheme is $K \cdot M^2$ for a particular ingress ER, where K is the total number of potential egress ERs, each having its own unique set of identifiers. We assume the admitted flow will send a TEARDOWN message to the ingress ER when it terminates. The TEARDOWN message contains the Fid, and its allocated bandwidth. Upon receiving the TEARDOWN message, the RM updates the FR accordingly and releases the Fid.

In the example shown in Figure 1, a new flow that arrives at ingress router ER_s is assigned an Fid with the first subfield, $FidIn$ equals to x, and $FidEg$ equals y. At ER_s, the new flow is aggregated with other flows with the same subfield, $FidIn$ = x, for group policing. At ER_d, this flow is grouped with other flows with $FidEg$ = y for policing. Every flow will be policed at both its ingress and egress ER in two distinct groups, thereby increasing the chances of detecting misbehaving flows. Every ER maintains only the aggregate state for each group and hence does not store any per-flow state.

3.1.2 Explicitly Assigned vs User-Selected Fids

In our approach, we attempt to maximize the level of flow aggregation without compromising the uniqueness of $Fids$, thereby minimizing the number of groups to be policed at every edge router(ER). Explicit assignment has two distinct advantages: First, the amount of aggregate policing needed to be performed at the router can drastically be reduced. For example, if there are n flows from an ingress ER to a specific egress ER, an optimal assignment of explicit $Fids$ can be achieved by maintaining only $\sqrt{n}$ groups at each of the two routers. We discuss this optimal assignment in the next section. Secondly, flows with similar bandwidth requirements can be aggregated into a common group to increase the effectiveness of group policing. It reduces the chances of a small bandwidth misbehaving flow hiding in an aggregate containing some large bandwidth flows. This is because it becomes hard to distinguish minor variations of a large bandwidth flow from large-scale bandwidth violations of a small bandwidth flow.

On the other hand, if flows were allowed to assume their own $Fids$, then it would be necessary to maintain a membership function to map the random $Fids$ to a particular group. The cost of performing an online grouping of flows based on these functions would be very high because the $Fids$ are continuously changing. Techniques like Stochastic Fair Blue (SFB) [18] that use random hash functions cannot be applied because they do not provide an inverse mapping from group identifiers to actual $Fids$. Thus, using SFB alone does not provide a direct mechanism to verify whether a suspected flow is truly misbehaving.

3.1.3 Other Challenges

Routing Changes: When a routing change occurs and causes an active flow to change its ingress or egress points, the previously assigned Fid may not match the group identifiers in one of the new ERs or both. Whenever this happens, we can either

1. remark the packets of this flow as best-effort, or
2. contact the RM for re-admission of this flow with the new endpoints.

Further investigation is needed to understand the performance and security issues of both approaches.

Meeting Fid Demands: The typical number of simultaneous flows observed on an ingress link is between 300 and 5000 [19]. It is expected that 5-10% of these will be latency sensitive applications and need Fids. Even if the demand for Fids increases 10 times in future, we need at most $M = \sqrt{5000}$, which is roughly 71 unique identifiers per $\mathcal{A}$ set. Based on the discussion in [32], the total number of ERs within an ISP, K, is in the order of 500.[3] Therefore, the total number of unique identifiers required for the whole ISP ($M \times K$) is roughly 35,500 in this case. Allocating 16-bits ($2^{16} = 65536$) for each subfield should be more than sufficient for producing mutually disjoint $\mathcal{A}_i$ for all routers.

Security Concerns with identifiers: One important security concern with our Fid assignment is the notion of *bogus identifiers*.

[3] K=number of POPs $\times$ number of ERs/pop. Major Tier-1 ISPs has about 25 POPs in the continental USA and each POP consists of a few core routers and 10-20 gateway routers.

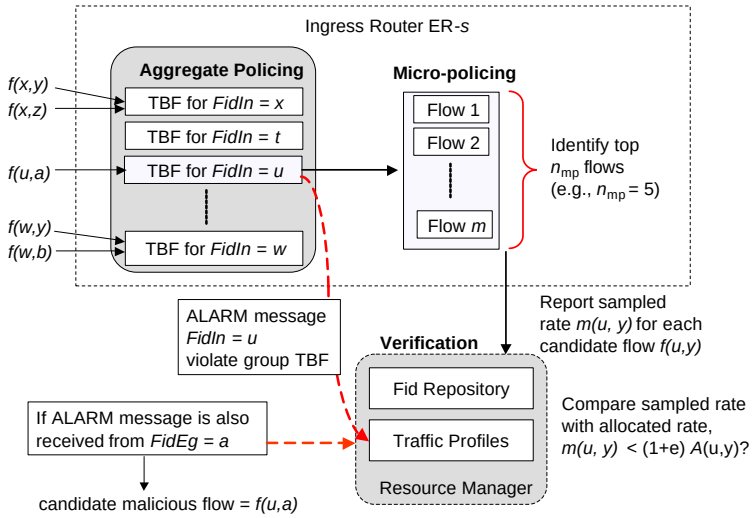

Figure 2: Control blocks of DCAP.

An external misbehaving user/application may attempt to create its own *bogus* Fid which is valid but has not been explicitly assigned by the RM and generate a flow with such an identifier. One possible way of dealing with bogus identifiers is to use the notion of one-way trapdoor functions. A function $f()$ is said to be a one-way trapdoor function if it is one-one and easy to compute while $f^{-1}()$ should be very hard to compute unless given a *trapdoor* [20]. The classic example of a one-way trapdoor function is public-key cryptography. Let us assume that the RM and the ERs within one ISP know a one-way function $f()$ with its corresponding trapdoor, t (not known to the external world). If a flow is supposed to be allocated an identifier, $Fid = FidIn, FidEg$, the actual identifier explicitly assigned to it will be $f(Fid)$ as opposed to just Fid. Upon receiving a packet with a flow identifier, x, the edge router first computes $f^{-1}(x)$ to find the corresponding ingress/egress group identifiers. Since one-way trapdoor functions are very hard to break, this can prevent the problem of bogus identifiers. To avoid replay attacks (using previously specified identifiers), we can just continuously modify the group identifiers at all edge routers. However, there are two basic problems with this approach: (a) computing f^{-1} may sometimes be an expensive operation especially if public-keys are used. (b) 32-bits in a flow identifier may be insufficient to perform these operations. A detailed analysis of security concerns of our approach is outside the scope of this paper.

3.2 DCAP: Detection via Collaborative Aggregate Policing

DCAP is mainly designed for detecting a small number of misbehaving flows among a large group of admitted flows. There are three stages in the detection process:

- Perform aggregate policing at both ingress and egress routers to quickly identify a subgroup that contains the misbehaving flows,
- Guess candidate flows within the group that violate aggregate bandwidth allocation via sampling within a specified time window, and
- Verify whether these flows are truly misbehaving.

Figure 2 highlights the major control blocks of DCAP. The remaining of this section discusses the technical details of each detection phase and example pseudo-codes are included in Figure 3.

3.2.1 *Aggregate Policing*

DCAP deploys traffic policer (TPs) at both ingress and egress ERs for policing the traffic from admitted flows with the matching group identifiers ($FidIn$ or $FidEg$). Each TP is a collection of continuous-state Token Bucket Filters (TBF) [9, 36, 30]). A TBF consists of a counter, which is incremented by the size of each arriving packet, and decrements periodically by a specific rate, as long as the value of the counter is positive. Every group identifier, $x \in \mathcal{A}_i$, is associated with a TBF with two parameters, r_{tot} and b_{tot}, where r_{tot} is the total average rate of admitted flows and b_{tot} is the total burst size. When a new flow between ER_s and ER_d with allocated bandwidth A_{new} is admitted and assigned an Fid, the RM updates the TPs at both ingress and egress routers. The total acceptable rate r_{tot} for the TBF with the matching $FidIn$ and $FidEg$ (at ER_s and ER_d, respectively) is increased by A_{new}. Packets that violate the associated traffic profile are discarded. Each TP keeps track of the dropped packets and reports the statistics to the RM.

Since the policing at the TP is performed on a group of flows sharing the same 16-bit subfield of Fid, the amount of state information maintained at the ingress ER is proportional to M, the number of unique identifiers in the set, $\mathcal{A}$. Consider an example ISP domain with K edge routers and M=100. Each ER maintains only 100 pieces of state information, but an arbitrary router can admit up to $K \times 10,000$ flows with unique Fids. A per-flow policing scheme would have require each ER to maintain all $K \times 10,000$ states.

3.2.2 *Providing Best Guesses*

```
// pseudo-code excuted by ingress router
PROCEDURE ADC(RECV)
//called when a RECV message arrives
    (A, s ,d) = read(RECV)
    // get the allocated rate, A, ingress and egress point, s and d
    if (measured load < bottleneck link capacity):
        (FidIn, FidEg) = get_Fid()
        // get unique Fid from RM
        r_tot(FidIn) = r_tot(FidIn) + A
        //update TBF w/ FidIn with allocated rate, A
        update(d, A)
        //contact egress router d, to update TBF w/ FidEg

// pseudo-code executed by ingress and egress router
PROCEDURE MONITOR()
    for each k in TBF_LIST:
        if tot_arrival[k] > r_tot[k]:
            send_ALARM(identifier[k])
            MICRO(k)
            //if TBF k overflows, send ALARM to RM
            //enter micro-policing mode

PROCEDURE MICRO(k)
    for each flow in TBF[k]:
        m[flow] =  sample_rate
    list = find_top5_flows()
    //list[flow]=(Fid of flow, measured rate of flow)
    send_RM(list)

// pseudo-code executed by the Resource Manager
PROCEDURE VERIFY(list)
    for each flow in list:
        if m_flow > (1 + e) A_flow:
            penalize(flow)
```

Figure 3: Pseudo-codes for DCAP mechanisms.

As an example, let a flow with $Fid = [f, g]$ be misbehaving. All flows with the same $FidIn = f$ will be policed as an aggregate in the same Token Bucket at the ingress ER, TP_s, regardless of what their $FidEg$ is. If the total allocated rate of $FidIn = f$ is violated, the affected TP reports f to the RM using an ALARM message (Figure 2). However, this information alone is insufficient for pinpointing the exact misbehaving flow, because there can be as many as $K \cdot M$ flows with the same $FidIn$ that could potentially be misbehaving. If the TP at an egress ER $FidEg = g$ also sends an ALARM message, the RM guesses that a flow with Fid that contains both f and g as its subfield is misbehaving, and submits this Fid for verification.

However, a misbehaving flow may not always be detected at both its ingress and egress ERs. To improve the effectiveness of DCAP, we introduce a "micro-sampling" mode. Whenever a group TBF that violates the aggregate rate is identified, DCAP requires the edge routers to sample individual flows within this group for a duration of t_{mp}. At the end of this period, the associated ER identifies n_{mp} largest flows, and report their Fids, along with the sampled rate, to the RM. Normally the potentially misbehaving flows are the ones that transmit at a much higher rate relative to other members.

In this paper, we consider t_{mp}= 5 seconds and n_{mp}= 5 for performance evaluation. With the assumption that the number of misbehaving flows is relatively small, n_{mp}=5 is reasonable, as shown in the ns simulations (Section 5). However, fixing a value of n_{mp} does have its disadvantage. It limits the efficacy of the micro-policing in two scenarios: (a) if the number of large flows is greater than n_{mp}, and (b) if many small misbehaving flows are hiding in the aggregate after discounting the n_{mp} large flows. An alternative solution is to report all the flows that are "disproportionally" larger than the rest in the group. We can leverage the fact that flows with *similar bandwidth* are aggregated for policing in the same group to compute the relative "size" of the flows as the following. Let r_{tot} be the allocated bandwidth to a traffic aggregate that has n admitted flows. We estimate the bandwidth requirement of individual flows as r_{tot}/n. All the flows with the sampled rate exceeding r_{tot}/n by a large margin (say 5%) are potentially misbehaving and will be reported to RM.

3.2.3 Verifying Misbehaving Behavior

For each reported flow with $Fid = [x, y]$, the RM compares the allocated rate, $A(x, y)$ with the measured rate $m(x, y)$ reported in the ALARM message:

$$m(x, y) < (1 + \epsilon) \cdot A(x, y) \quad (1)$$

where ϵ is a hysteresis parameter to absorb transient behavior of bursty traffic. If the condition in (1) is violated, the flow is considered misbehaving. ϵ is typically between 0 and 0.05. A counter associated with this flow is incremented for every such violation of condition (1). To reduce the probability of false alarm, we introduce a second hysteresis parameter, η, which determines the minimum number of violations before a flow is reported as misbehaving. A reasonable range for η is between one and five.

3.2.4 Hiding in the Aggregate

Although group policing allows our architecture to scale, it limits the effectiveness of DCAP because a misbehaving flow can "hide" in the aggregate. This can happen when:

- the aggregate usage of the group is less than the total allocated rate because certain flows are under-utilizing their resources, and
- The misbehaving flow is relatively small compared to other larger, yet legitimate, flows in a misbehaving group.

To address this problem, we introduce redundancy by deploying a traffic policer at every egress point as well. By assigning a unique Fid to every flow, we ensure that no two flows are in the same group in both the associated ingress and egress ERs. Essentially every flow is policed in the aggregate at two distinct points to maximize the number of misbehaving flows that are detected. Secondly, we assign flows with similar bandwidth requirement into the same group (Property 3 of Fids in Section 3.1.1).

3.3 Other Issues

3.3.1 Implementation of the Resource Manager

Although the Resource Manager (RM) is described as a single logical entity within a domain, it can be implemented as a distributed architecture across POPs. Every POP of an ISP usually has a fault monitoring facility to continuously manage the link and router status in its network. The additional functions of the RM can be implemented as additional pieces of software that reside in these monitoring facilities. For example, a *local-RM* of a POP can maintain part of the domain's database, by tracking $Fids$ where at least one of the $FidIn$ or $FidEg$ is a valid group identifier of an edge router within the same POP. For every flow that is admitted, a new entry with its Fid and allocated bandwidth is created in the partial FR databases at both its ingress and egress POPs. Similarly, when a flow stops, we remove the flow's entry from the local RMs in its ingress and egress POPs, respectively.

An alternative model for building a Resource Manager would be to just distribute the state amongst the edge routers as opposed to aggregating it at specific monitoring points within a POP. This alternative model is mainly applicable outside the realm of ISPs where we cannot assume the presence of any POP structure. (refer to section 4.2, for the applicability of this model and DCAP for overlay networks)

While the alternative model may no longer have the state reduction property of DCAP (i.e. the nodes will maintain per-flow state), the nodes still need to perform only aggregate policing. This is because, the *active state* at a node (state used for policing flows on a per-packet basis) is still the aggregate state while the per-flow state is accessed less frequently.

3.3.2 Penalizing Misbehaving Flows

Once a misbehaving flow is detected, several penalty actions can be taken against it: e.g., dropping all the packets of this flow, demoting all its packets to best-effort, or charging more for the connection. Such a penalty would require keeping some state information at the edge router, but only for a very small subset of flows that misbehave (assuming the number of misbehaving flows is small). Our framework can support any of these penalty actions, but for simplicity, we choose packet dropping as the default. We will not address the impact of different penalty actions in detail, since the main focus of this paper is the design of a scalable detection system to identify and isolate these malicious flows. We also do not consider the additional state incurred for containing the malicious flows after they are identified.

4. APPLICABILITY OF DCAP

In the previous section, we described our solution to the malicious flow detection within one ISP's network. In this section, we will describe different scenarios under which our solution can be applied. In particular, we will consider 3 specific scenarios: (a) Single ISP scenario, (b) Policing in Overlay Networks, (c) End-to-end flows traversing multiple ISPs.

Until now, we have been vague about the definition of a flow. The notion of a *flow* in our work can be more generic than just an end-to-end connection between two end-hosts. DCAP places three constraints on the definition of a flow:

1. Every flow is associated with a unique flow-identifier as specified by an ISP (refer to Section 3.1 for more details). At a router, all packets with the same flow-identifier belong to one flow.
2. All packets of a flow are associated with the same ingress and egress routers within the ISP's network.
3. If multiple end-to-end connections have the same QoS requirements and share the same intra-domain path between a specific pair of ingress and egress routers, they can be treated as a single flow if the connections originate from the same *customer* that is accountable for billing purposes (from an ISP's perspective).

4.1 Single ISP Scenario

We will now use the generality in the definition of a flow to show how DCAP can be used for policing within an ISP. Service Level Agreements(SLAs) offered by one ISP to another is normally a complicated agreement, which is carefully worded to address different aspects of performance(latency, bandwidth, loss) at a very coarse granularity. A sample SLA document of a popular ISP is available online [3]. These SLAs are normally static and negotiated on a monthly/yearly basis rather than on very small time granularities.

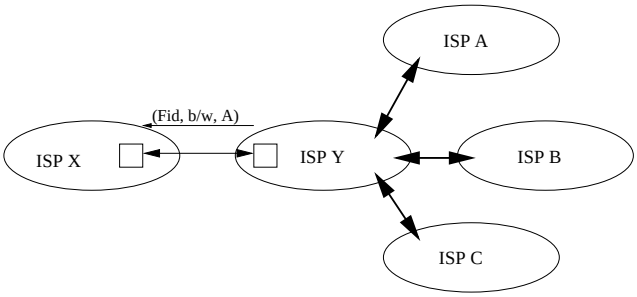

Figure 4: Supporting Dynamic SLAs between ISPs

An ISP can offer to provide transit service between two of its neighboring ISPs. In the simplest case, the transit ISP statically provisions a certain amount of bandwidth between its two neighbors. DCAP provides a scalable way of extending this notion of SLAs to a dynamic reservation model within one ISP without making much modifications to the existing Internet architecture. Figure 4 illustrates a simple scenario where ISP Y has four neighbors A,B,C,X. ISP Y can dynamically provision a certain amount of bandwidth b from ISP X to ISP A and present the edge router of ISP X with an Fid and a corresponding bandwidth b. ISP X can potentially insert this Fid to any packet destined for ISP A (which can be directly determined from the BGP tables in ISP X[4]). Similarly, ISP Y can potentially offer many such $Fids$ of varying bandwidths dynamically to ISP X for each of its other neighbors A,B,C.

DCAP provides a scalable policing model to ensure that the neighboring ISPs do not violate their SLA agreements. However one may think that such a system can be deployed by simply using a per-flow or a single aggregate policing model. We argue that DCAP provides two distinct advantages over both cases in this scenario. First, certain tier-1 ISPs have around $N = 500 - 2000$ ISPs as neighbors. In the worst case scenario, if each pair of these ISPs ($N \times N - 1$) setup an SLA through their common neighbor, per-flow policing may not be a scalable option. Second, between a pair of neighboring ISPs(say X and A), it may be beneficial for ISP Y to handle multiple $Fids$ of varying bandwidths rather than a single Fid of a certain aggregate rate. This offers ISP X the additional flexibility of partitioning these $Fids$ offered by Y amongst its different customers.

4.2 Overlay QoS Provider Scenario

OverQoS [34] and Service Overlay Networks [13] are two recent proposals to provide some form of end-to-end QoS guarantees using an overlay network. These architectures enable a third-party to set up an overlay network and offer customers coarse-level bandwidth guarantees. Traffic policing is also an integral component of these architectures.

The fundamental reason why DCAP would be appropriate for such overlay architectures stems from the assumption that the entire over-

[4]If Y peers with A at multiple locations, the traffic from X to A needs to be diverted along a single egress point of Y to preserve the semantics of a flow.

lay is owned by a single administrator. To deploy DCAP, we assume that every overlay node could be a potential egress point and the Fid allocation is based on the ingress and egress points (entry and exit points within the overlay) of a QoS flow. Here, a flow could either represent a single end-to-end connection or an aggregate pipe for a customer. However, we make one assumption: The ingress and egress overlay node remain the same during the course of a flow. This assumption definitely holds for at least the ingress node since a typical deployment strategy for many overlay-based solutions is to enable the first overlay node as the default gateway of an end-host.

An overlay network may not have specific aggregation points to provide the functionality of a resource manager for a collection of nodes. Hence, this functionality needs to be distributed across all overlay nodes (as described in Section 3.3.1).

4.3 QoS across Multiple ISPs

In order to use DCAP as the traffic policing building block for providing QoS across multiple ISPs, we need to make a few assumptions. First, we require an explicit RSVP-like admission control phase to signal an individual flow's performance requirements to the ingress nodes of all ISPs. The ISPs along the path can locally decide to admit or reject the flow based on the availability of its network resources. Second, an admitted flow is supposed to carry a stack of identifiers (one Fid per ISP along the path). Once a packet of a flow exits an ISP's boundary, the egress router should strip off the corresponding Fid from the top of the stack. Third, when a connection ends, the flow sends its ingress router an explicit message (TEARDOWN) to release the resources.

These assumptions also pose as challenges towards deploying DCAP in a multiple ISP scenario to provide end-to-end QoS. IntServ [11], having been proposed almost a decade ago is far from being deployed. The presence of multiple ISPs raises a fundamental limitation towards deploying end-to-end QoS.

An alternative aspect currently under investigation is whether SLAs between ISPs could be composed together for providing some meaning form of end-to-end QoS guarantees. In Section 4.1, we presented a model of a dynamic SLA that one ISP acts as a transit provider and offers a pipe abstraction between two of its neighboring ISPs. Consider a simple scenario of 4 ISPs, A,B,C and D in a routing path. Assume that dynamic SLAs between A-B-C (i.e. B offering a transit SLA between A and C) and B-C-D with identifiers f_1 and f_2 are already in place. We can compose these identifiers at the transit point between B and C to offer a pipe abstraction between A and D. If such a model can be extended end-to-end, we can potentially offer coarse bandwidth guarantees between various pairs of stub networks. If such a pipe abstraction between stub networks is associated with multiple $Fids$ the stub networks is responsible for allocating these $Fids$ to the individual flows.

5. PERFORMANCE EVALUATION

In this Section, we evaluate the performance and robustness of DCAP via simulation study. We first describe our methodology in Section 5.2, including details on network topology and traffic generation in our simulations. In Section 5.3, we list the four test scenarios, which are designed to address specific issues. The results and detailed discussions of each case are presented in Section 5.4-5.5.

5.1 Simulation Model: Assumptions

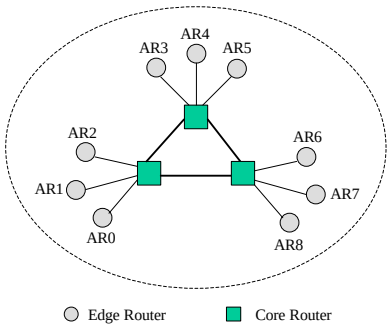

Figure 5: Simulation Topology

To distinguish between QoS flows and non-QoS flows, we introduce two classes of flows: *high-priority* and *best-effort*. We assume that high-priority flows require statistical QoS guarantees while best-effort traffic does not receive any performance assurance. Core routers employ priority queuing to distinguish QoS flows from non-QoS flows. We use a per-flow reservation protocol like RSVP [8] to signal individual flow's performance requirements to the ingress nodes, which locally admit or reject new flows based on the availability of the network resources.

There are typically two types of admission control mechanisms: *parameter-based* or *measurement-based*. Measurement-based admission control (MBAC) algorithms base their decisions on measurements of existing traffic while parameter-based techniques admit flows based on worst-case bounds of different quality metrics (delay, bandwidth, loss rate). Therefore, MBACs are best suited for providing soft real-time service without hard guarantees. Many MBAC algorithms and principles outlined in [22, 23, 21, 24] can be applied in combination with DCAP. For our simulation study and lab prototype reported in this paper, we use the Measured Sum (MS) algorithm: a new flow arriving at edge router ER_s and destined for edge router ER_d with bandwidth requirement A_i is admitted if:

$$A_i + L_{\text{tot}}(s,d) \leq R(s,d)$$

where $L_{\text{tot}}(s,d)$ is the total estimated traffic and $R(s,d)$ is the total capacity reserved for carrying high priority traffic between any pair of ingress-egress routers, (s,d). $R(s,d)$ could be statically configured by an ISP based on traffic predictions or dynamically allocated based on aggregate reservation requests. We assume the former case in this paper.

5.2 Simulation Setup

We use the ns-network simulator to implement the basic mechanisms of DCAP. The TP[5] is implemented as a connector in front of a node, and a time-window estimator is introduced at each input link to estimate the rate of existing flows. The admission control module is created as an NsObject and inserted before the ingress ERs. The various tasks of the RM in our architecture are implemented at the Tcl-level. Our DCAP-patch works for ns-2.1b6.

Since it is infeasible to run large-scale Internet simulations over ac-

[5]We modify the DiffServ module contributed by Sean Murphy, http://www.teltec.dcu.ie/ murphys/ns-work/diffserv/index.html.

tual networks, we use ns to simulate a simple subgraph of an Internet topology shown in Figure 5 that consists of a set of core routers (CRs) and egress routers (ERs). The CRs are fully meshed, while the ERs form stub networks connected to individual CRs. Flows from host networks enter and exit the network domain through edge routers, AR0-AR8, where they are aggregated for policing using the DCAP scheme. All routers support priority scheduling and there is enough buffering for 200 packets at each queue. All control signaling between the RM (not shown on the figure) and the ERs is carried in UDP messages.

We are unable to access real traffic traces from Internet backbone networks because such data is proprietary. Instead, we derive traffic models based on published results and data sets collected by ISPs themselves.

The arrival process of the admission-controlled traffic is modeled as Poisson with arrival denoted as $\lambda_i(t)$. We use the index t to indicate the time-of-day dependence of the traffic demand as reported in [16] and [19]. For example, the bandwidth consumption typically peaks between 10 a.m. and 2 p.m. during the day and shows a dip from midnight to 3-4 a.m. To reflect the realistic traffic demand, we introduce $\pm$ 10-15% changes to $\lambda_i(t)$ at a regular interval of 30 minutes. The traffic distributions from an ingress ER to a set of egress ERs are based on a random probabilistic model.

We use four kinds of traffic source models in our experiments:

1. We use EXP1 to model a typical Voice-over-IP source. EXP1 has exponential on and off times with an average of 1.004 s and 1.587 s, respectively. This corresponds to a 38.53% talk-spurt cycle, as recommended by ITU-T specification for conversational speech [5]. The peak transmission rate is 64 kbps, and the average is 24.8 kbps.
2. EXP2 also has exponential on and off times, but with an average of 100 ms and 900 ms, respectively. The peak rate is increased to 248 kbps while keeping the average rate the same as EXP1, leading to a burstier source.
3. CBR is a constant bit rate source of 64 kbps.
4. PARETO source has Pareto on and off times but the average rate is the same as EXP1.

EXP1, EXP2 and CBR have exponential lifetimes with an average of 300s. The flow lifetimes of PARETO sources follow a lognormal distribution with average of 300 s. The aggregation of Pareto sources is known to exhibit long-range dependencies [37]. For all four cases, packets are 320 bytes in length.

In our simulations, we assume that each new flow will request for bandwidth r that is equal to its average rate, regardless of its source model. The network does not have a priori knowledge on the peak rate or characteristics of the flows.

5.3 Test Scenarios

As discussed in Section 2.1, there are different performance criteria in evaluating DCAP as a misbehaving flow detection mechanism. In particular, we are interested in two events: successful detection and false alarms. The probability of successful detection $P_{\rm sd}$ is approximated as the fraction of misbehaving flows that are actually detected. Similarly, the probability of false alarm $P_{\rm fa}$ is the fraction of normal flows that are incorrectly reported as misbehaving. Since the flows are policed as an aggregate, misbehaving flows can cause packets from complying flows to be dropped. The probability of a packet being incorrectly dropped, denoted as $P_{\rm mis}$, quantifies the impact of misbehaving flows on end-to-end performance seen by other member flows.

We study the trade-offs between these different metrics by tuning the parameters of DCAP. Following the discussion in Section 2.1, it is more desirable to sometimes miss the detection of a misbehaving flow (hence lower $P_{\rm sd}$) than to wrongly penalize the good flows and hurt their performance. Therefore, the objective of DCAP is to maximize $P_{\rm sd}$, while keeping $P_{\rm fa}$ and $P_{\rm mis}$ to near zero.

To examine the robustness of DCAP, we simulate four extreme cases:

Case 1: False Positive Detection
This is a control experiment with zero misbehaving flows to determine the optimum choice of DCAP parameters to reduce the number of false alarms.

Case 2: Homogeneous Flows:
This arrangement is similar to Case 1, but now a small fraction, γ, of the flows are misbehaving. All the flows generate traffic with the same average rate and have similar characteristics.

Case 3: Mixing Elephants and Mice
Flows generally do not have similar bandwidth requirements in a real network. Previous studies show that 20% of the flows (known as elephants) contribute to 80% of the Internet traffic. It is important to understand how DCAP copes with such scenario.

In this test scenario, we assign Fids such that one large flow (the elephant) and many simultaneous small flows (mice) are grouped together for policing. The allocated rate of the elephant flow $A_{\rm l}$ is 10 times larger than the allocated rate of a small flow, $A_{\rm s}$. All of the small flows are compliant, and only the large flow misbehaves.

Case 4: Hiding in the Aggregate
Again, we consider a mixture of one large flow and many simultaneous flows like Case 3. However, the large flow is compliant this time, and a fraction γ of the small flows are misbehaving. This is the case in which the small misbehaving flows may survive the group policing without being detected by hiding in the group aggregate (Section 3.2.4) if we only perform regular sampling.

We repeat each experiment using four different source models: EXP1, EXP2, CBR and PARETO. For each scenario, the simulation was repeated 10 times with different random seeds, and the average $P_{\rm sd}$, $P_{\rm fa}$, and $P_{\rm mis}$ was computed. Each simulation ran for 1000s. All experiments were performed under high load with 20% blocking probability. A misbehaving source requests allocation for r kbps but sends traffic at a higher rate, randomly chosen between $1.1 \cdot r$ and $1.2 \cdot r$ kbps (10-20% violation). The average and peak rate for each source model is the same as described in Section 5.2.

5.4 Results and Discussions

Case 1: False Positive Detection

The experiments in Case 1 are intended for understanding the limitations of the DCAP and its performance sensitivity with respect to different choices of design parameters. Ideally, none of the flows should be reported as "misbehaving", but the transient behavior of

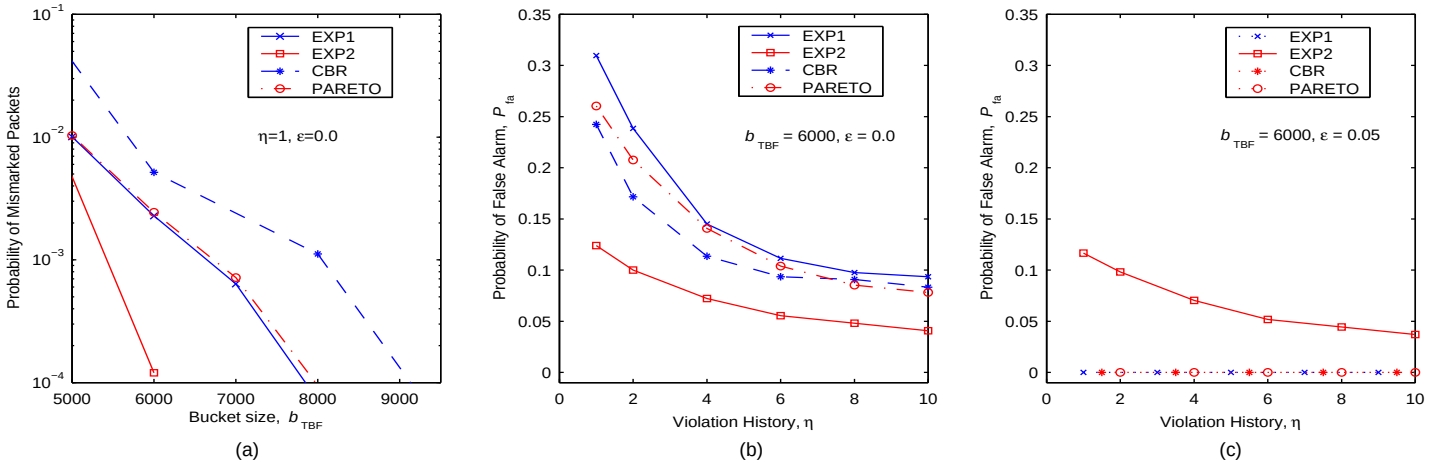

Figure 6: Case 1: Zero Misbehaving Flows.

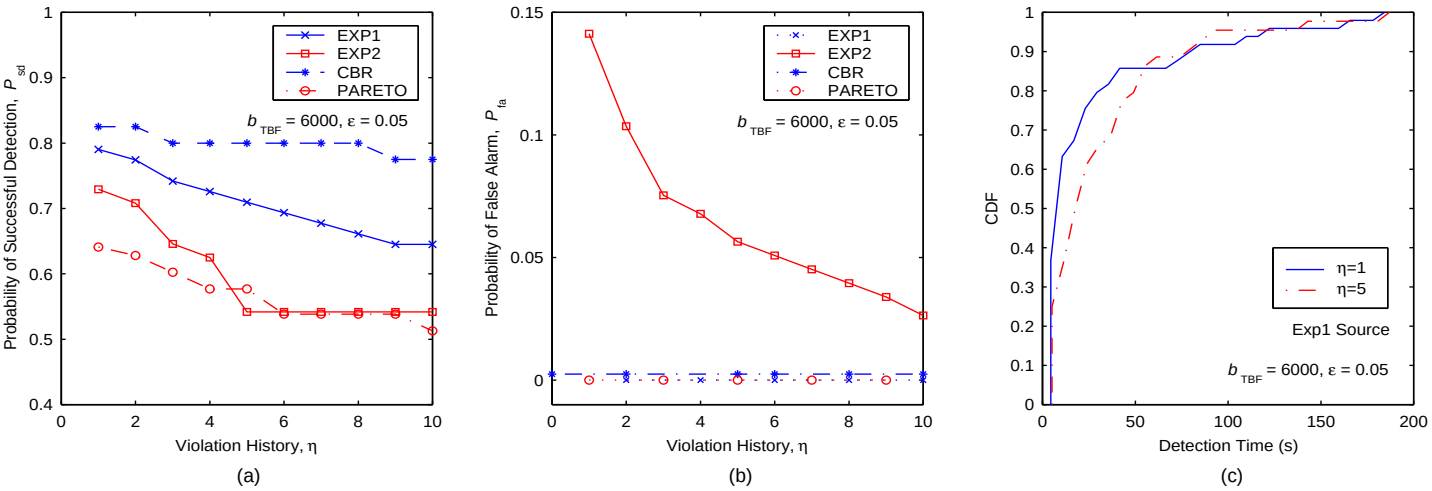

Figure 7: Case 2: Many small homogeneous flows; a small fraction, γ =0.1, misbehave.

bursty traffic could momentarily overflow the Token Bucket Filters (TBFs) and be interpreted as misbehaving, leading to a "false alarm". Figure 6a shows how the choice of bucket size, $b_{\rm TBF}$ at the Traffic Policers (TP) affects the probability of mis-marked packets, $P_{\rm mis}$. A smaller value of $b_{\rm TBF}$ is more effective in detecting misbehaving flows, but there should be enough tokens to allow the legitimate packets to pass, and keep the $P_{\rm mis}$ low. Except for the CBR traffic, $P_{\rm mis}$ is below 0.01 for other source models. When the $b_{\rm TBF}$ is set to 6000 using a fixed leaky rate, all the flows (including CBR traffic) can be policed with losses $P_{\rm mis}$ less than 0.005. The two hysteresis parameters η and ϵ determine under what conditions a flow is reported as "misbehaving" (Section 3.2.3), but have no effect on $P_{\rm mis}$.

We can relax the condition for DCAP by increasing ϵ and η, and this helps to reduce the number of false alarms. Figure 6b and 6c study how $P_{\rm fa}$ varies as a function of η for ϵ = 0.0 and 0.05. For a 0% tolerance level in DCAP (i.e., ϵ=0), $P_{\rm fa}$ decreases gradually as η is increased. However, we notice that $P_{\rm fa}$ drastically decreases for all the source models when the tolerance level is increased to 5%. This indicates that $P_{\rm fa}$ is more sensitive to ϵ than η. For the rest of the experiments, we choose ϵ=0.05 and $b_{\rm TBF}$ = 6000.

Case 2: Homogeneous Flows Scenario

Increasing η causes a delay in reporting misbehaving flows and may adversely impact the effectiveness of DCAP. We examine this issue in Case 2. We set the value of γ (fraction of misbehaving flows) to be 0.1. Figure 7a and 7b show the variation of $P_{\rm sd}$ and $P_{\rm fa}$ as η is increased from 1 to 10. The effect of η on $P_{\rm sd}$ for the CBR source is minimal. For the other source models, $P_{\rm sd}$ decreases sharply when η is increased and the rate of decrease varies across the source models. From Figure 7b, we can infer that only in the case of the EXP2 source model is $P_{\rm fa}$ sensitive to the value of η. With η = 1, we can detect most of the misbehaving flows with EXP1 (79%), CBR (83%) and PARETO (64%) sources with virtually zero $P_{\rm fa}$. In the case of EXP2, there is a trade-off between maximizing $P_{\rm sd}$ and minimizing $P_{\rm fa}$ as we choose the value for η. This indicates that burstier sources are more difficult to detect. The observed $P_{\rm mis}$ is between 0.02-0.79%.

We also measured the detection time for each correctly identified misbehaving flow and plotted the distributions in Figure 7c. With

Table 1: Case 3: One large misbehaving flow and many small complying flows. η = 5, b_{TBF} = 6000, ϵ = 0.05.

Source	EXP1	EXP2	CBR	PARETO
P_{sd}	1.0	1.0	1.0	1.0
P_{fa}	0.0077	0.027	0.012	0.0013
P_{mis}	0.003	0.00021	0.0072	0.0037

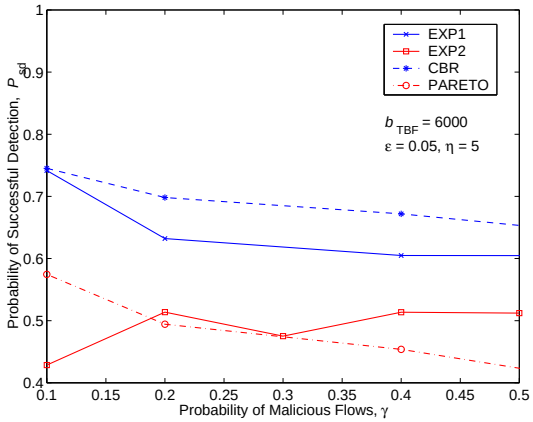

Figure 8: Case 4: One large flow (A_{l}) and many small flows (A_{s}); γ of small flows misbehave.

η = 1, the average detection time is 26.9 seconds, which is less than 1/10 of the average duration of a flow. 90% of the flows are detected within 78.9 seconds. When we increase η to 5, the average detection time increases to 33.8 seconds, which is still reasonably fast. The 90^{th}-percentile detection time is 80.4 seconds in this case.

The simulations in Case 2 show the basic results of DCAP hold across different source models. Although long range dependent traffic like PARETO is harder to detect, we can achieve a reasonable success rate (0.64) with zero false alarms. The presence of burstier sources, EXP2, pose challenges to the DCAP scheme, and we need to choose the value for η carefully to maximize P_{sd} while keeping P_{fa} reasonably small.

Case 3: Mixing Elephant and Mice

We have so far considered only homogeneous flows with the same rates. In the next two cases, we consider an extreme case where one large flow and many small flows are aggregated together for policing (Section 5.3). We repeat our experiments for four different source models. In Case 3, only the large flow is misbehaving. The results are summarized in Table 1. For all source models, we always successfully detect the misbehaving large flow and P_{fa} is at most 0.027.

Case 4: Hiding in the Aggregate

Case 4 addresses the scenario where misbehaving small flows "hide" in the aggregate with another large flow. The probability of a small flow being misbehaving is γ. Intuitively, we suspect that detection is harder in this case, because the misbehaving flows can "steal" the idle bandwidth allocated to the large flow. Since the traffic policer can only enforce the total allocated rate, the misbehaving flows may not be detected. Figure 8 shows the P_{sd} achieved for different values of γ and Table 2 summarize P_{sd}, P_{fa} and P_{mis} for γ = 0.1 and 0.5. Surprisingly, we notice that the P_{sd} achieved with γ = 0.1 for

Table 2: Case 4: One large flow and many small flows. γ of small flows misbehave. η = 5, b_{TBF} = 6000, ϵ = 0.05.

Source Model	EXP1	EXP2	CBR	PARETO
$\gamma = 0.1$				
P_{sd}	0.74	0.43	0.75	0.57
P_{fa}	0.00066	0.011	0.0	0.0
P_{mis}	0.0032	0.00016	0.0047	0.0028
$\gamma = 0.5$				
P_{sd}	0.61	0.51	0.67	0.39
P_{fa}	0.00067	0.025	0.0	0.0
P_{mis}	0.0030	0.00047	0.0088	0.0022

EXP1, CBR, and PARETO sources are fairly close to the results in Case 2, where there is no large flow. But for EXP2, the success rate is significantly smaller (P_{sd}=0.43 in Case 4 as supposed to 0.54 in Case 1). When γ increases, the success rate P_{sd} decreases for EXP1, CBR and PARETO source models. With EXP2, P_{sd} fluctuates as we vary γ, and is actually higher at γ=0.5 than γ=0.1. This is because the active cycle of EXP2 is very short (10%), and can easily go undetected if it coincides with the idle period of the large flow. However, when the fraction of misbehaving flows increases, there is an increased likelihood that some of the misbehaving flows will synchronize or overlap in their active cycles, leading to overflow of the TBF at the traffic policer. When the aggregate rate is violated, all the flows sharing the same subfield ($FidIn$ or $FidEg$) will be monitored individually (micro-monitoring) and the misbehaving flow can be correctly identified. The probability of false alarms P_{fa} and mis-marked packets P_{mis} are negligible in this case across different values of γ and source models.

5.5 Further Sensitivity Analysis

So far, we have been considering flows with homogeneous source characteristics in our simulations. The next experiment uses a random mixture of the four different source models (EXP1, EXP2, CBR and PARETO), each with different peak rates, idle times and burst times. Each arriving flow chooses among these source models at random. We repeat the experiment in Case 2, with η=1 using heterogeneous flows (HET), and compare the results with Case 2 where homogeneous flows are used. Results are summarized in Table 3. With HET sources, the success rate P_{sd} is lower than all the other homogeneous source models, but the differences in P_{fa} and P_{mis} are negligible. It is difficult to tune the hysteresis or TBF parameters to optimize the overall performance since the source characteristics are not known a priori.

We repeat the Case 2 experiment using the EXP1 source with the following modifications:

a. DCAP without micro-policing mode, and

b. DCAP deployed at ingress ERs only.

Results show that only 23% of misbehaving flows are detected in (a), and 53% in (b), which is significantly lower than 79%, when DCAP is deployed at both ingress and egress ERs (Figure 7).

6. IMPLEMENTATION

In this section, we provide a brief description of a DCAP prototype implemented on top of the Click modular router [25]. We use this implementation to evaluate certain performance metrics

which could not be accurately quantified through simulations. One such important metric is the overhead incurred at an edge router by adding DCAP control functionalities. The current implementation works on Linux 2.2.16 and 2.2.17 kernels.

6.1 Overview of Prototype

Click is a Linux-based software router, and is assembled from packet processing modules called elements. Individual elements implement simple router functions like packet classification, queuing and scheduling. We extend the Click router to support three additional elements: the traffic policing (TP) unit, the reservation agent(RA), and DCAP agent. The RA is responsible for directing flow requests to the Resource Manager (RM) and forwarding responses from the RM to the client. The TP is implemented as a set of token bucket filters to police admitted flows. When a specific group of aggregate traffic exceeds its allocated threshold, an alarm is sent to the DCAP agent, which then handles the micro flow policing and verification process to identify the misbehaving flows as described in Section 3.2.

The communication between the Click router and the Resource Manager is performed through SNMP. In order to enhance the throughput of the Click router, we reduce the number of context switches required for processing the control packets from the RM by batching the messages from the RM to the Click router.

6.2 Experimental Setup

Using our DCAP prototype, we measure the performance overhead of adding the Reservation and Monitoring agents in an edge router. To quantify this overhead, we compare the maximum throughput obtained from our implementation to a basic implementation of Click which did not contain any of the monitoring tools (hereafter referred to as default Click). This experiment helps provide insights as to whether it is practical to deploy DCAP.

For evaluation purposes, we set up our own cluster of machines over a 10.0.0.0/24 network. The throughput measurements depend on the configuration of the network testbed and the machine implementing the Click router. For studying the performance, we restrict ourselves to the communication between the RM and one edge router (implemented on top of Click). The machine running the Click router has the following configuration: Pentium-III 650 Mhz, 3com 3c905 Ethernet controller. We use one other machine with a similar configuration as the RM. We used four other machines as sources and sinks of traffic. All these machines are connected to a backbone router using 100 Mbps connections. The router is a Bay Networks Accelar-1100B router with the capacity to support 16 100 Mbps Ethernet ports. The traffic statistics was periodically sent to the RM every 100ms. We modified Mgen [2], a publicly available traffic modeling software, to generate traffic for our experiments.

Table 3: Comparisons between heterogeneous and homogeneous source models: γ = 0.1, b_{TBF} = 6000,ϵ = 0.05.

Source Model	HET	EXP1	EXP2	CBR	PARETO
$\eta = 1$ P_{sd}	0.55	0.79	0.73	0.83	0.64
P_{fa}	0.0	0.0	0.14	0.0025	0.0
P_{mis}	0.0030	0.0028	0.00016	0.0079	0.0031

Table 4: Average Performance Reduction vs number of flows

Number of flows	Reduction in Throughput(%)
≤ 300	≤ 0.1
350	1.5
400	2.8
450	1.6

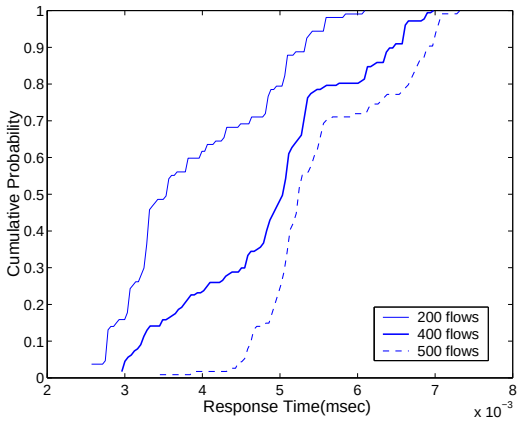

Figure 9: Response Time of Flow Allocation for varying loads

6.3 Experimental Results

In our first experiment, we measured the maximum throughput of our implementation at different loads and compared it to a default Click router. A basic flow in our setup has a bandwidth of 80 kbps and a packet size of 1024 bytes. As the number of flows increases, the amount of policing and state needed at the edge router also increases.

Table 4 shows the average reduction in the throughput of the access router for varying number of flows. We make two observations. First, the overhead incurred due to DCAP, i.e., the percentage of throughput degradation from default Click, is small. The maximum throughput difference observed in all our experiments is 5% and the average degradation is much smaller than this. Second, we notice that the throughput of the Click router saturates in our system at around 400 flows. Beyond 500 flows, we observed packet drops in the network interface.

In the second experiment, we measured the response time for flow allocation at different loads. We achieved a particular load in the system by maintaining a constant number of active flows and sending dummy flow requests to the Click router from the Traffic generator. There are three stages in the process of obtaining a response for a flow request: the RA in Click forwards the request to the RM, the RM performs admission control on the flow, and the RM sends the response to the requesting entity through the RA.

In Figure 9, we plot the cumulative distribution of the response time at three different loads: 200 flows (small load), 400 flows (saturation point) and 500 flows (high load). From the graph, we can observe that the mean response time increases as the load increases and the CDF shifts to the right. In all our experiments, we observe a minimum response time of 2.5 ms and a maximum of 7.2 ms. Our results indicate that the standard deviation of our response time is high. This can be attributed to the batching of responses at the RM and timer-based processing of flow requests at the Click

router. However, this variance is tolerable since the mean response time is small.

7. CONCLUSIONS

Although detection of misbehaving flows has been recognized as an important aspect of resource control, a practical and scalable way of implementing it has not been studied in great detail. This paper proposes a new scheme called DCAP (Detection via Collaborative Aggregate Policing) for policing incoming flows and detecting misbehaving behavior without requiring per-flow state maintenance at any edge routers. By aggregating flows for group policing, DCAP only requires $O(\sqrt{n})$ state maintenance at edge routers (where n is the number of flows), which is substantially better than previous approaches. Extensive simulations show that DCAP is effective and robust across a variety of source models and extreme cases. For the EXP1 source (VoIP type), DCAP can successfully detect 79% of misbehaving flows with almost zero false alarms and about 0.3% incorrectly-dropped packets. However, further study is needed to improve the detection of bursty misbehaving sources. Our approach has significant practical value since DCAP can be implemented with simple per-packet operations in a high-speed line card on a router. Our prototype demonstrates that DCAP adds minimal processing overhead to edge routers.

7.1 Future Work

As part of the future work, we will address the various practical issues of deploying DCAP in the existing Internet.

Changes to Routers: To deploy DCAP, no changes are required in the core routers, while the policing and monitoring need to be added to the edge routers. From our Click implementation, we infer that the modifications needed to add the extra DCAP mechanisms in an edge router is minimal. The entire implementation consists of 4463 lines of C++ code. Per-packet operations of DCAP can also be implemented in hardware.

Alternate Accounting Mechanism: In this paper, we perform regular sampling during the "micro-policing" phase of DCAP to track the bandwidth usage of individual flows within a sub-group. In future, we will consider alternative accounting mechanisms, including the scheme proposed in [14], to track the top flows (also known as heavy hitters) that contribute the most to the traffic within a group.

Security Issues: In Section 3.1.3, we briefly specified some of the security concerns (like bogus identifiers) with our solution. While we have made some in-roads towards addressing some of these concerns, a more detailed analysis of security issues is part of future work.

8. ACKNOWLEDGMENTS

We are grateful to Jennifer Rexford, Scott Shenker, Eddie Kohler, and Ion Stoica for their insightful feedback, and to Nina Taft, Supratik Bhattacharyya, and Dina Papaqianaki for sharing their knowledge on current IP-backbone topology and traffic models.

9. ADDITIONAL AUTHORS

Additional authors: George J. Lee (Massachusetts Institute of Technology - Department of Electrical Engineering & Computer Science, email: `gjl@mit.edu`).

10. REFERENCES

[1] Cisco Netflow. http://www.cisco.com/warp/public/732/Tech/netflow/.

[2] The Naval Research Laboratory (NRL), Multi-Generator (MGEN) toolset. http://manimac.itd.nrl.navy.mil/MGEN/.

[3] Verio service level agreement: Terms and definitions. http://www.verio.com/company/policies/sla/tsandcs.cfm.

[4] ITU-T: Recommendation I. 371, Traffic control and congestion control in b-isdn, 1993.

[5] ITU-T Recommendation P.59, Artificial conversational speech, 1993.

[6] M. Basseville and I. V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*. Prentice Hall, April 1993.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weise. An architecture for differentiated services. RFC 2475, IETF, December 1998.

[8] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Reservation protocol (RSVP) version 1 functional specification. RFC 2205, IETF, September 1997.

[9] M. Butto, E. Caverolla, and A. Tonietti. Effectiveness of the leaky bucket policing mechanism in atm networks. *IEEE J. Selected Areas in Communications*, 9(3):335–342, April 1991.

[10] C.-N. Chuah, L. Subramanian, R. H. Katz, and A. D. Joseph. Qos provisioning using a clearing house architecture. In *Proc. of IFIP 5th Intl. Workshop on Quality of Service*, pages 115–124, June 2000.

[11] D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proc. ACM SIGCOMM*, August 1992.

[12] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. *Internetworking: Research and Experience*, 1:3–26, January 1990.

[13] Z. Duan, Z. L. Zhang, and Y. T. Hou. Service overlay networks: Sla, qos and bandwidth provisioning. In *Proc. International Conference on Network Protocols*, November 2002.

[14] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[15] K. C. F. Baker and and A. Smith. Management information base for the diffserv architecture. Internet draft, IETF, May 2002. draft-ietf-diffserv-mib-16.txt.

[16] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational ip networks: Methodology and experience. *IEEE/ACM Trans. Networking*, 9(3):265–279, June 2001.

[17] W. Feng, D. Kandlur, D. Saha, and K. Shin. Stochastic fair blue: A queue management algorithm for enforcing fairness. In *Proc. IEEE INFOCOM*, April 2001.

[18] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. Blue: A new class of active queue management algorithms. In *Proc. IEEE INFOCOM*, April 1999.

[19] C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, K. Papagiannaki, and F. Tobagi. Design and deployment of a passive monitoring architecture. In *Passive and Active Measurement Workshop*, April 2001.

[20] S. Goldwasser and M. Bellare. Lecture notes on cryptography. http://www.cs.ucsd.edu/users/mihir/papers/gb.html.

[21] M. Grossglauser and D. Tse. A framework for robust measurement-based admission control. In *Proc. ACM SIGCOMM*, pages 237–248, September 1997.

[22] S. Jamin, P. Danzig, S. Shenker, and L. Zhang. A measurement-based admission control algorithm for integrated services packet networks. *IEEE/ACM Trans. Networking*, 5(1):56–70, February 1997.

[23] S. Jamin, S. Shenker, and P. Danzig. Comparison of measurement-based admission control algorithms for controlled-load service. In *Proc. IEEE INFOCOM*, April 1997.

[24] E. W. Knightly and J. Qiu. Measurement-based admission control with aggregate traffic envelopes. *IEEE/ACM Trans. Networking*, 9(2):199–210, April 2001.

[25] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Trans. on Computer Systems*, 18(4), November 2000.

[26] S. Machiraju, M. Seshadri, and I. Stoica. A scalable and robust solution for bandwidth allocation. In *Proc. of IFIP 5th Intl. Workshop on Quality of Service*, May 2002.

[27] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Trans. Networking*, 1(3), June 1993.

[28] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. *IEEE/ACM Trans. Networking*, 2(2), April 1994.

[29] E. P. Rathgeb. Modeling and performance comparison of policing mechanisms for atm networks. *IEEE J. Selected Areas in Communications*, 9(3):325–334, April 1991.

[30] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. RFC 2212, IETF, September 1997.

[31] S. Shenker and J. Wroclawski. General characterization parameters for integrated service network elements. RFC 2215, IETF, September 1997.

[32] A. Sridharan, S. Bhattacharyya, C. Diot, R. Guerin, J. Jetcheva, and N. Taft. On the impact of traffic aggregation on routing performance. In *Proc. of Intl Teletraffic Congress*, September 2001.

[33] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queuing: Achieving approximately fair bandwidth allocations in high speed networks. In *Proc. ACM SIGCOMM*, pages 118–130, September 1998.

[34] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz. Overqos: Offering qos using overlays. In *First Workshop on Hot Topics in Networking(HotNets)*, October 2002.

[35] B. Tietelbaum and R. Geib. Internet2 qbone: A test bed for differentiated services. In *Proc. of INET*, June 1999.

[36] J. S. Turner. Managing bandwidth in atm networks with bursty traffic. *IEEE Network*, pages 50–58, September 1992.

[37] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-similarity through high-variability: Statistical analysis of ethernet lan traffic at the source level. In *Proc. ACM SIGCOMM*, pages 100–113, August 1995.